Formalising the CHISEL Feature Notation

Ken Turner
University of Stirling

19th May 2000

Introduction

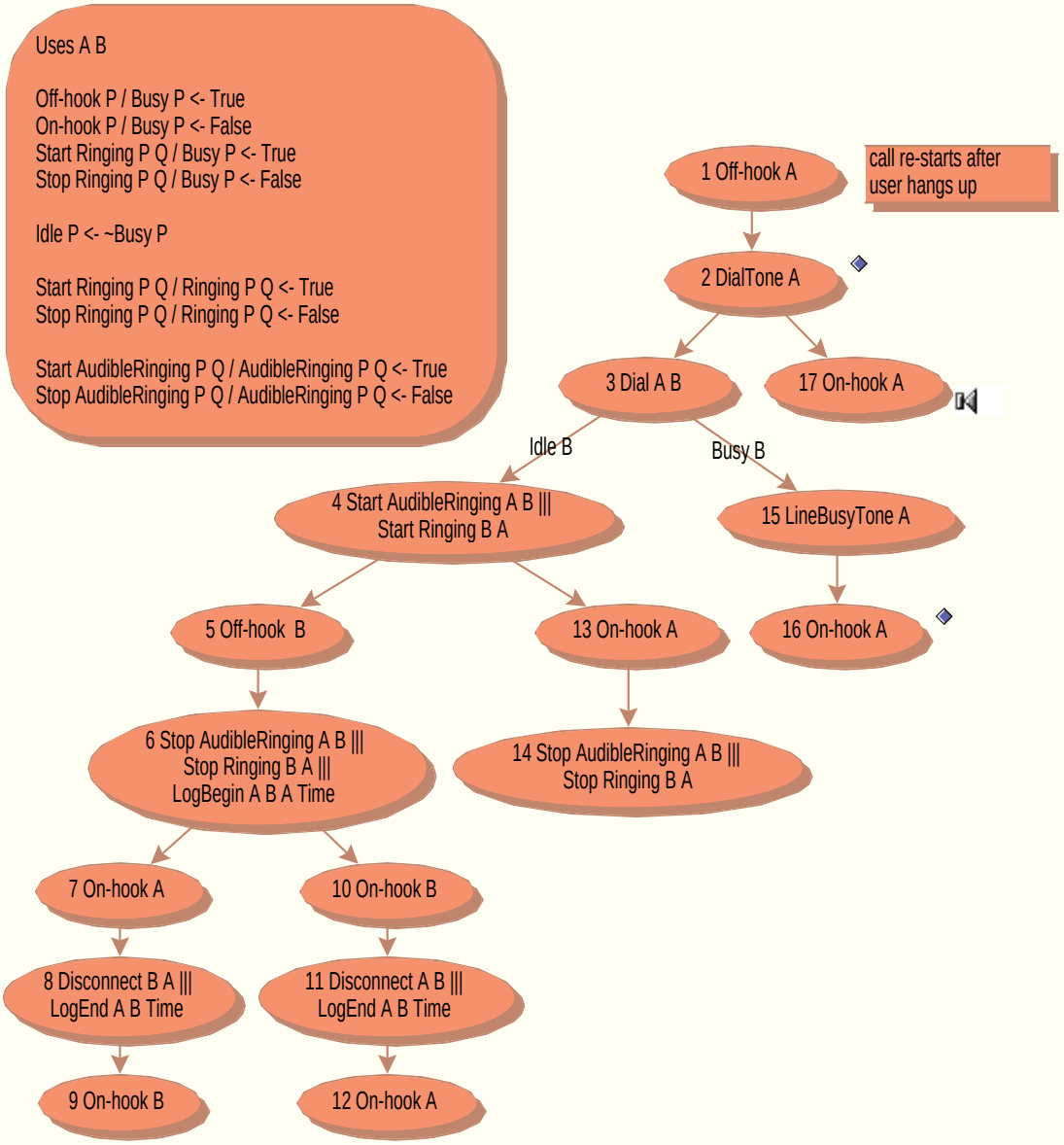
- CHISEL defined by BellCore (Telcordia) as a graphical notation for services
- CRESS (CHISEL Representation Employing Systematic Specification):
  - simplifies CHISEL diagrams
  - extends CHISEL capabilities
  - formalises CHISEL interpretation
  - offers open tool support
  - may be used with various diagram editors and formal methods
- CRESS capabilities:
  - syntax and static semantics enforced
  - use of formal language hidden
  - standard interaction detection methods
  - automatic translation to SDL, LOTOS
  - automated analysis via SDL, LOTOS
  - so far, mainly focused on feature representation and formalisation

Motivation

- attractions of CHISEL:
  - reflects industrial practice
  - simple and accessible notation
  - self-contained feature descriptions
  - deals with a variety of feature types
- aims of CRESS:
  - extend CHISEL for greater usability
  - retain backwards compatibility
  - define diagram rules more precisely
  - integrate explanations as formal rules
  - formalise interpretation of diagrams
  - provide an open toolset
  - support adoption of CHISEL

Root Diagram – POTS

- a root diagram is the base for features
- the order of input/output events is defined
- events may happen in parallel
- guards can be used to control behaviour
- Plain Old Telephone Service in CRESS:



Root Diagram – CRESS Extensions

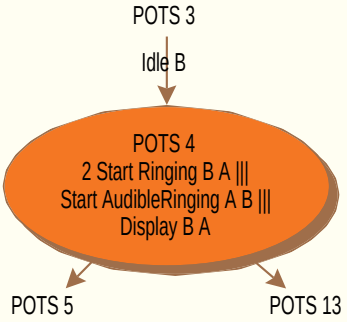
- a rule box formalises:
  - feature variables:
    Uses A B
  - interdependency among features:
    Uses C / POTS CFBL
  - setting status variables:
    StartRinging P Q / Busy P <– True
  - macro-like functions:
    Idle P <– ~Busy P
- comment attachments may be any file
- a **Start** node may be introduced for diagrams with loops
- redundant diagram details optional:

  POTS 4 B<–B A<–A

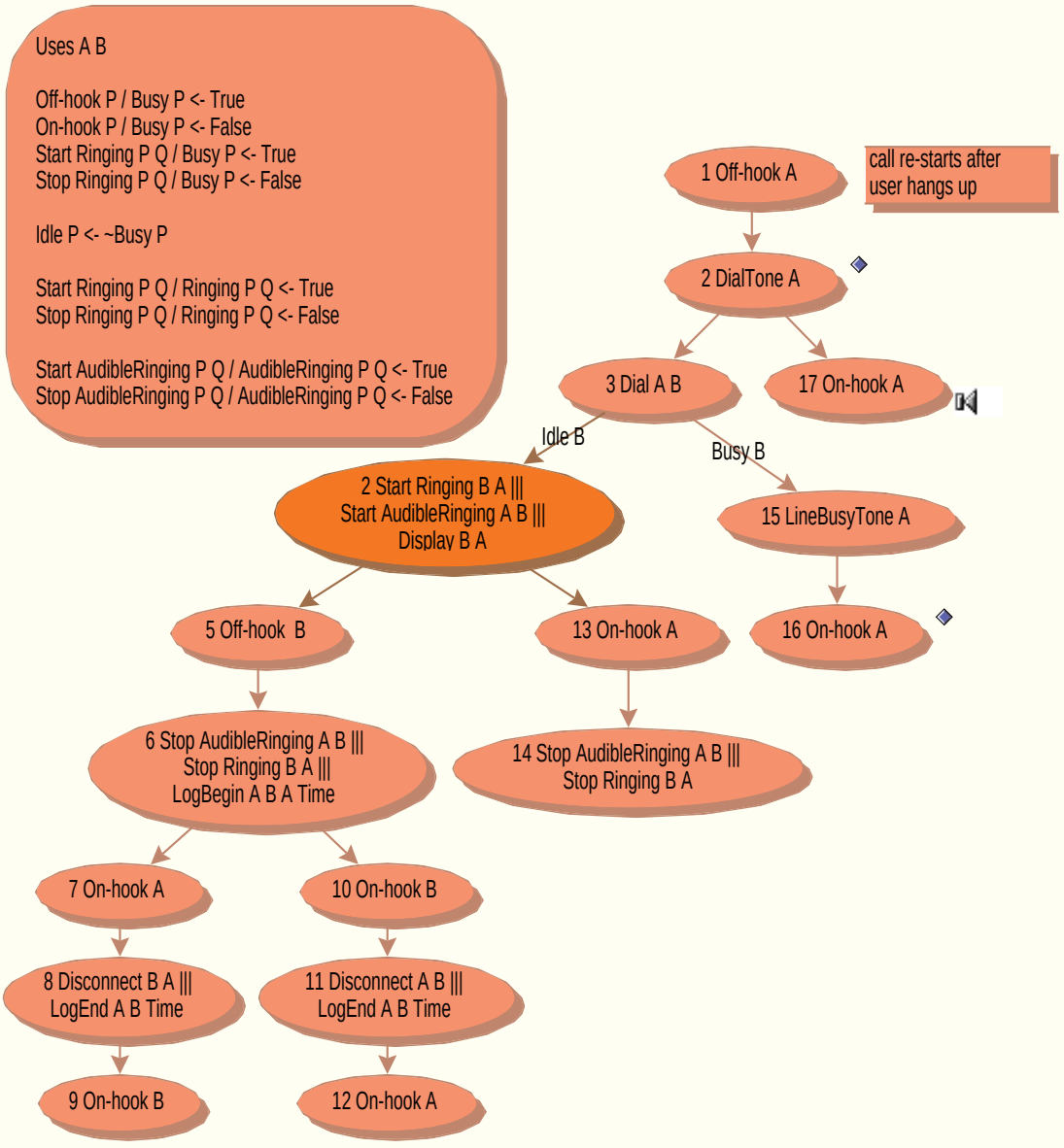
  could be just POTS 4
- expression operators are formalised, including conditional expressions
- an **Else** guard may be used

Feature Diagram – CND

- a feature diagram modifies the root
- Calling Number Delivery:



is spliced into POTS:



Feature Diagram – CRESS Extensions

- feature composition rules formalised:
  - addition of behaviour or guard
  - replacement of behaviour
  - deletion of behaviour
- diagram references simplified:

  SELF 3 B<–B A<–C

  becomes just 3 A<–C
- top-level root node may be extended
- composition rules clarified:
  - source node binding read backwards:

    POTS 4 B<–C A<–B

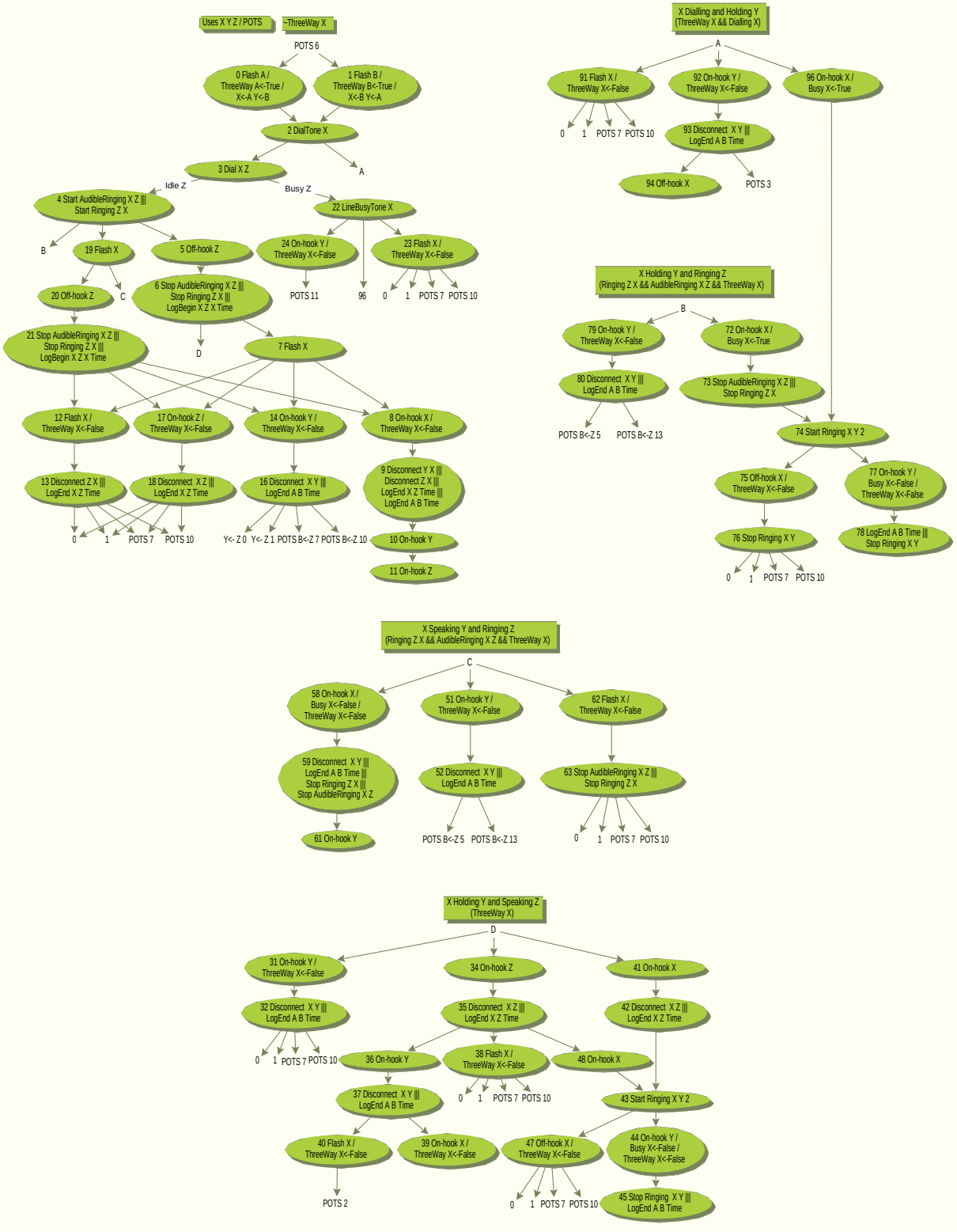
    uses B for C, A for B in a feature
  - target node binding read forwards:

    POTS 4 B<–C A<–B

    uses C for B, B for A in the root
  - consistency checked statically

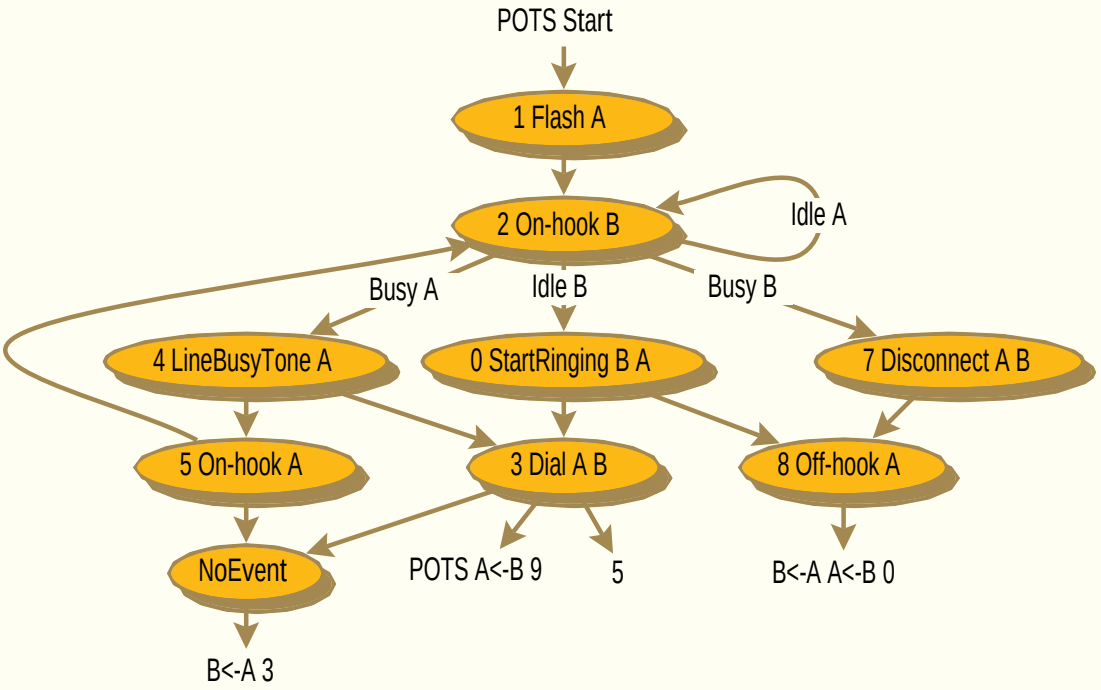
Feature Diagram – TWC

- Three-Way Calling uses linked diagrams:



Feature Diagram Loops

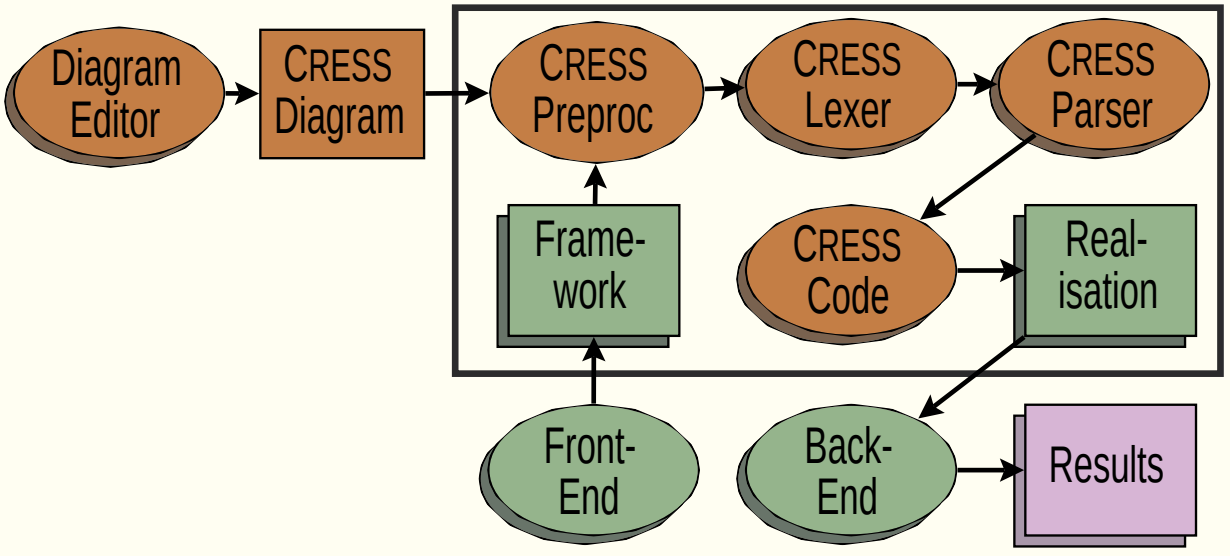
- loops arise from:
  - self-loops
  - links via other diagram nodes
  - **NoEvent** as a short-hand empty node
- complex (in)direct loops may thus arise:



- a cyclic graph must hence be handled:
  - graph building is harder than for a tree
  - a **Start** node may be needed
  - a revisited node is not followed further
  - but must be analysed when revisited

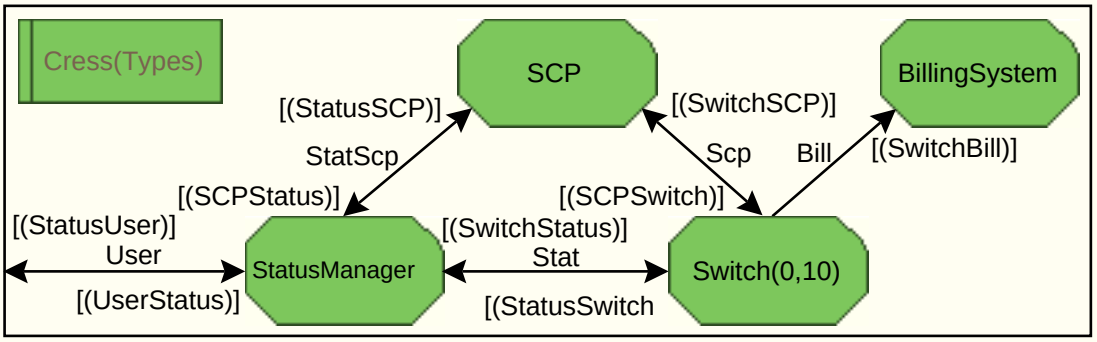
Translating CRESS

- diagrams are prepared with any reasonable editor (e.g. Diagram!)
- diagrams are automatically combined
- a pre-defined specification framework is chosen (e.g. SDL, LOTOS)
- the composite diagram is translated automatically to the chosen language
- the translated diagram is inserted into specification framework
- the result is simulated, prototyped, analysed, validated, ...
- CRESS tools are portable and extensible
- toolset architecture:



SDL Representation

- specification framework:



- translation strategy:

CRESS	SDL
input	**State** *Diagram.Node*; **Input** *Signal(Parameters)*; Diagram.Node:
output	*Diagram.Node*: **Output** *Signal(Parameters)*;
guard	**Decision** *Guard*; (True): *Behaviour* (Else): ... **EndDecision**;

- SDL input rules cause major problems:
  - input must be preceded by a state
  - same signal in same state only once
  - merging branches must repeat input

LOTOS Representation

- specification framework:

```
Specification Network [User] : NoExit
  Cress(Types)
    Behaviour Hide Bill,Stat,Scp In
       ((StatusManager [Stat] |[Stat]|
          SCP [Scp,Stat]) ||| BillingSystem [Bill])
      ||
       Cress(CFBL,CND,INCF,TWC)
    Process BillingSystem [Bill] : NoExit ...
    Process StatusManager [Stat] : NoExit ...
    Process SCP [Scp,Stat] : NoExit ...
  EndSpec
```

- translation strategy:

CRESS	LOTOS
input	User !*Signal* ?*Parameters* User !*Signal* !*Parameters*
output	User !*Signal* !*Parameters*
guard	[*Guard*]–> ...

- the main complications are:
  - input must know if parameters defined
  - status variables read as required
  - common process for merging branches

Feature Validation

- most effort to date on formalising CHISEL
- standard techniques can be used:
  - simulation
  - state exploration
  - observers
  - watchdogs
- SDL can use automated:
  - exhaustive and other state exploration
  - MSC-based validation of features in isolation or in combination
- LOTOS can use automated:
  - test expansion
  - test generation

Conclusion

- CRESS (CHISEL Representation Employing Systematic Specification):
  - simplifies CHISEL diagrams
  - extends CHISEL capabilities
  - formalises CHISEL interpretation
  - offers open tool support
- CRESS results:
  - syntax and static semantics enforced
  - automatic translation to SDL, LOTOS
  - translation better than manual efforts
  - standard interaction detection methods